Fast automatic type class derivation

with shapeless

Who

Alexandre Archambault

Software engineer / "type B" data scientist

`@alxarchambault`, `github.com/alexarchambault`

PhD

Worked at

» Small prop. trading firm, back-test / assess trading algorithms
» Since Sept. '15: *Teads*

Teads.tv

Native video advertising: video ads inside articles

Reach 100M unique users / day, half that 6 months earlier

Machine learning

modelisation

case class / sealed trait intensive

```scala
case class WebPage(url: String, categories: Seq[Category])

sealed trait Category
case object Politics extends Category
case object Food extends Category
```

10s – 100+ in big projects

Type classes

- convert to/from JSON
- persist in a binary format
- render them in CSV (if possible)
- automatically generate them
- pretty-print
- any kind of marshalling or reification
- ...

Automate their generation!

macros?

Scala type system is quite complex

- type parameters, path-dependent types, local types, accessibility, ...

Surprising things in their API

- side-effects!

type class derivation with shapeless

Printer

```scala
trait Printer[T] {
  def apply(t: T): String
}
```

Helper method

```scala
def print[T](t: T)(implicit printer: Printer[T]): Unit =
  println(printer(t))
```

Goal

```
> print(WebPage("https://news.com/food", Seq(Food)))
WebPage:
  https://news.com/food
  Seq:
    Food
```

HLists

Sequence of types / sequence of values

```
val empty: HNil = HNil
val oneLen: Int :: HNil = 1 :: empty // type: ::[Int, HNil]
val twoLen: String :: Int :: HNil = "b" :: oneLen // type: ::[String, ::[Int, HNil]]
// etc.
```

Induction!

Type classes for `HLists`

```scala
implicit def hnilPrinter: Printer[HNil] = Printer { l =>
  ""
}

implicit def hconsPrinter[H, T <: HList](implicit
  hp: Printer[H],
  tp: Printer[T]
): Printer[H :: T] = Printer { case h :: t =>
  hp(h) + "\n" + tp(t)
}
```

Induction
- rank 0
- rank n+1 if rank n

Printers for standard types

```scala
implicit val stringPrinter: Printer[String] = Printer { s => s }
implicit val intPrinter: Printer[Int] = Printer { n => n.toString }

implicit def seqPrinter[T](implicit tp: Printer[T]): Printer[Seq[T]] =
  Printer { seq =>
    if (seq.isEmpty)
      "Seq"
    else
      "Seq:\n" + indent(seq.map(tp(_)).mkString("\n"))
  }
```

Type classes for `HLists`

```
@ implicitly[Printer[Int :: String :: HNil]].apply(3 :: "ab" :: HNil)
res1: String = 3
ab



@ val p: Printer[Int :: String :: HNil] = hconsPrinter[Int, String :: HNil](
    intPrinter,
    hconsPrinter[String, HNil](
      stringPrinter,
      hnilPrinter
    )
  )

@ p(3 :: "ab" :: HNil)
res4: String = 3
ab
```

Reminder

```
implicit def hconsPrinter[H, T <: HList](implicit
  hp: Printer[H],
  tp: Printer[T]
): Printer[H :: T] = Printer { case h :: t =>
  hp(h) + "\n" + tp(t)
}
```

Generic
case classes <-> HLists

```
trait Generic[T] {
  type Repr
  def from(repr: Repr): T
  def to(t: T): Repr
}
```

Generic
case classes <-> HLists

```
// Example

@ val gen = Generic[WebPage]
res4: Generic[WebPage] {
  type Repr = String :: Seq[Category] :: HNil
} = $fresh$macro$78$1@6811bc30

@ gen.to(WebPage("https://news.com", Seq(Politics)))
res5: String :: Seq[Category] :: HNil = "https://news.com" :: Seq(Politics) :: HNil

@ gen.from("https://news.com" :: Seq(Politics) :: HNil)
res6: WebPage = WebPage("https://news.com", Seq(Politics))
```

Printer for case classes

```scala
implicit def genericPrinter[C, L <: HList](implicit
  gen: Generic.Aux[C, L],
  p: Printer[L]
): Printer[C] = Printer { c =>
  val name = c.getClass.getName
  val s = p(gen.to(c))
  if (s.isEmpty)
    name
  else
    name + ":\n" + indent(s)
}
```

```
@ print(WebPage("https://news.com", Seq(Politics)))
WebPage:
  https://news.com
  Seq:
    Politics
```

Not talking about sealed traits here

Printers for case classes

```
// What happens
val printer: Printer[WebPage] =
  genericPrinter[WebPage, String :: Seq[Category] :: HNil](
    Generic[WebPage],
    implicitly[Printer[String :: Seq[Category] :: HNil]]
  )
```

Wrong divergences

```
case class RankedWebPages(items: Seq[(WebPage, Int)])
```

```
@ print(RankedWebPages(Seq(WebPage("https://news.com", Nil) -> 2)))
Compilation Failed
diverging implicit expansion for type Printer[RankedWebPages]
starting with method hconsPrinter
print(RankedWebPages(Seq(WebPage("https://news.com", Nil) -> 2)))
     ^
```

SLS 7.2 😕: "complexity" should not increase

» complexity of `Printer[WebPage]` > complexity of `Printer[RankedWebPage]` but happens after

Wrong divergences

What we want:

```
genericPrinter(
  Generic[RankedWebPages],
  hconsPrinter(
    seqPrinter(
      genericPrinter(
        Generic[(WebPage, Int)],
        hconsPrinter(
          implicitly[Printer[WebPage]],
          hconsPrinter(
            intPrinter,
            hnilPrinter
          )
        )
      )
    ),
    hnilPrinter
  )
)
```

Recursive types

```
case class WebPage(url: String, categories: Seq[Category], parent: Option[WebPage] = None)
```

```
@ print(WebPage("https://news.com", Seq(Politics)))
Compilation Failed
diverging implicit expansion for type Printer[WebPage]
starting with method genericPrinter
print(WebPage("https://news.com", Seq(Politics)))
     ^
```

More robust derivation with Lazy

```scala
trait Lazy[T] {
  val value: T
}

implicit def hconsPrinter[H, T <: HList](implicit
  hp: Lazy[Printer[H]],
  tp: Lazy[Printer[T]]
): Printer[H :: T] = Printer { case h :: t =>
  hp.value(h) + "\n" + tp.value(t)
}

implicit def genericPrinter[C, L <: HList](implicit
  gen: Generic.Aux[C, L],
  p: Lazy[Printer[L]]
): Printer[C] = ...
```

Lazy

- Circumvents SLS 7.2 complexity issue: no more wrongly reported divergences
- Handles recursion (real ones)

Lazy

```scala
lazy val stringPrinter = implicitly[Printer[String]]
lazy val seqCategoryPrinter = implicitly[Printer[Seq[Category]]]

lazy val webPagePrinter: Printer[WebPage] = Printer { page =>
  val lines = Seq(
    "WebPage",
    stringPrinter(page.url),
    seqCategoryPrinter(page.categories)
  ) ++ page.parent.map(webPagePrinter(_))

  lines.mkString("\n")
}
```

```
case class Coordinates(lat: Double, long: Double)

object Coordinates {
  implicit val printer: Printer[Coordinates] = Printer { coord =>
    s"${coord.lat}:${coord.long}"
  }
}

@ Coordinates.printer(Coordinates(10.0, 20.0))
10.0:20.0


@ print(Coordinates(10.0, 20.0))
Coordinates:
  10.0
  20.0
```

:-(

Getting priorities right

```scala
trait DerivingPrinters {
  implicit val hnilPrinter: Printer[HNil] = ...
  implicit def hconsPrinter[H, T <: HList](implicit
    hp: Lazy[Printer[H]],
    tp: Lazy[Printer[T]]
  ): Printer[H :: T] = ...

  implicit def genericPrinter[C, L <: HList](implicit
    gen: Generic.Aux[C, L],
    p: Lazy[Printer[L]]
  ): Printer[C] = ...
}

object Printer extends DerivingPrinters {
  ...
}
```

Priorities: less invasive ways

- export-hook
  github.com/milessabin/export-hook
- or the right type class for that!
  github.com/alexarchambault/derive (branch)

```
implicit def genericPrinter[C, L <: HList](implicit
  ev: LowPriority[Printer[C]],
  gen: Generic.Aux[C, L],
  p: Lazy[Printer[L]]
): Printer[C] = ...
```

Make it faster

Benchmark

- Against macros
- Time to compile a large enough code base

upickle

- `Reader[T]`: has a `PartialFunction[Json, T]`
- `Writer[T]`: has a `T => Json`
- Nice test suite

`github.com/lihaoyi/upickle`

- default values
- change field names in JSON with annotations
- ...

Default

```
case class Element(
  id: String,
  categories: Seq[Category] = Seq(),
  location: Option[String] = None
)
```

```
@ Default[Element].apply()
res1: None.type :: Some[Seq[Category]] :: Some[Option[String]] :: HNil =
  None :: Some(Seq()) :: Some(None) :: HNil
```

Annotations

```scala
case class Element(
  @Name("ID") id: String,
  categories: Seq[Category] = Seq(),
  @Name("address") location: Option[String] = None
)
```

```
@ Annotations[Name, Element].apply()
res2: Some[Name] :: None.type :: Some[Name] =
  Some(Name("ID")) :: None :: Some(Name("address")) :: HNil
```

Annotation_

```scala
sealed trait Element

@Name("first") case class First(
  id: String,
  categories: Seq[Category] = Seq(),
  location: Option[String] = None
) extends Element


@ Annotation[Name, First].apply()
res3: Name = Name("first")
```

Benchmark

- type-level crazy
- shapeless 2.2
- shapeless 2.3 (Strict et al.)
- hybrid

`github.com/alexarchambault/upickle-pprint`

`github.com/alexarchambault/auto-type-class-benchmark`

Benchmark

type-level crazy

```
implicit def mkReader[T]
 (implicit
   priority: Lazy[Priority[
     Reader[T],
     Implicit[
       MkStdReader[T] :+:
       MkCoproductReader[T] :+:
       MkTupleReader[T] :+:
       MkProductReader[T] :+: CNil
     ]
   ]]
 ): Reader[T] = ...
```

Recursive type class `Implicit` to manually handle priorities

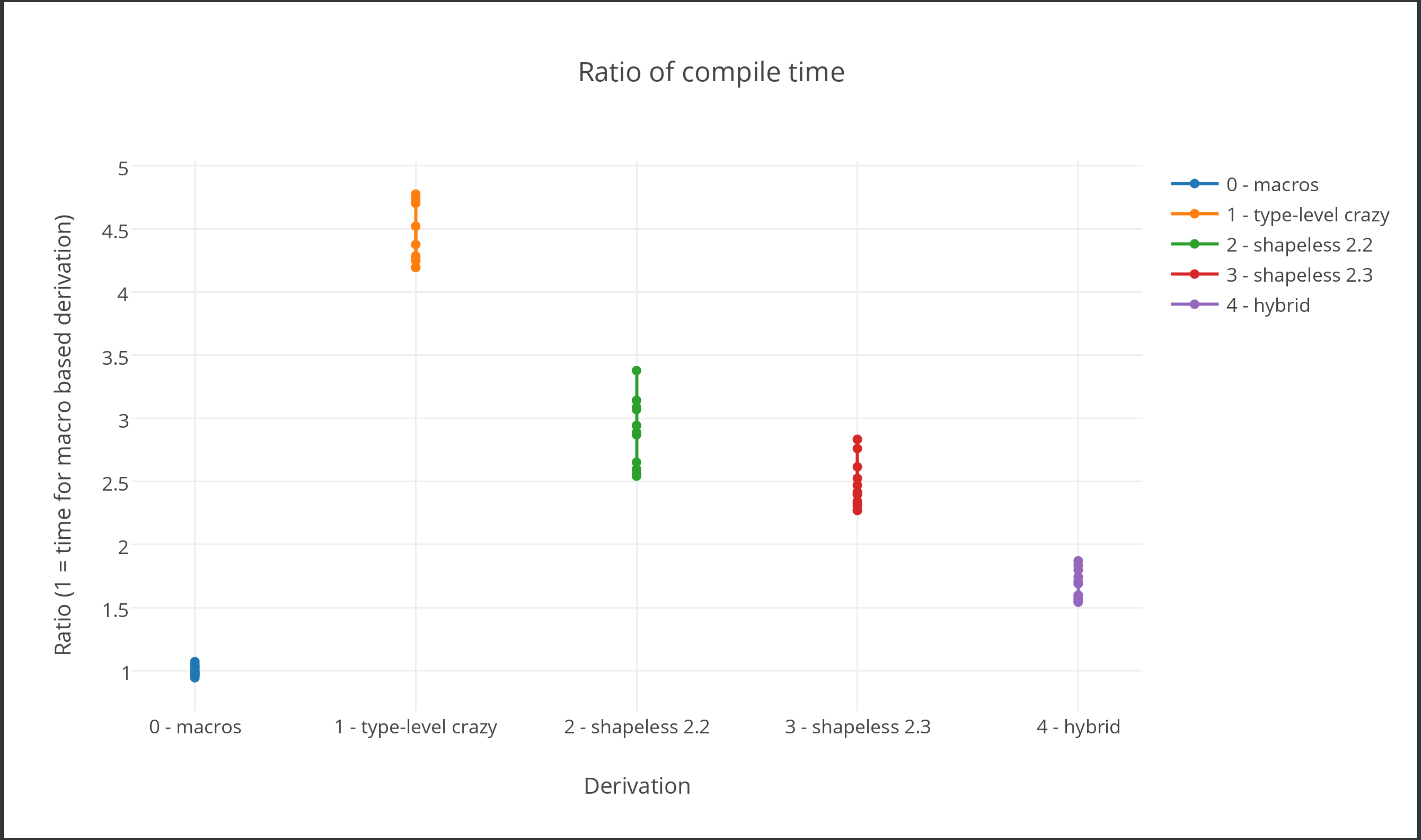
Benchmark
shapeless 2.2 / 2.3

```
object Lazy {
  def apply[T](t: => T): Lazy[T] =
    new Lazy[T] {
      lazy val value = t
    }
  ...
}

object Strict {
  def apply[T](t: T): Strict[T] =
    new Strict[T] {
      val value = t
    }
  ...
}
```

Benchmark hybrid

```scala
implicit def productReader[T](implicit
  reader: Derive[ProductReader[T]]
): Reader[T] = ...

object ProductReader {
  object typeClass {
    def point[A](a: => A): ProductReader[A] = ...
    def product[A, B](
      name: String, keyAnnotation: Option[String], default: => Option[A],
      head: => Reader[A], tail: ProductReader[B]
    ): ProductReader[(A, B)] = ...
    def map[A, B](underlying: => ProductReader[A], from: A => B): ProductReader[B] = ...
  }
}
```

github.com/alexarchambault/derive


Ratio of compile time
Ratio (1 = time for macro based derivation)
5
4.5
4
3.5
3
2.5
2
1.5
1
0 - macros
1 - type-level crazy
2 - shapeless 2.2
3 - shapeless 2.3
4 - hybrid
Derivation
0 - macros
1 - type-level crazy
2 - shapeless 2.2
3 - shapeless 2.3
4 - hybrid

Questions?

This presentation will be soon available on Skills Matter.com at the following link: https://skillsmatter.com/conferences/6862-scala-exchange-2015#skillscasts